UNIMA
CENTRO UNIVERSITÁRIO DE MACEIÓ | Afya

Especificação do Projeto: "OptiRota" - Um Sistema Avançado de Otimização de Rotas para Logística Urbana

Prof.Icaro Ferreira.

Sumário Executivo

Este documento delineia a especificação técnica e a estrutura de gestão de projetos para o "OptiRota", um sofisticado sistema de otimização de rotas concebido como um projeto avançado para uma disciplina de Estrutura de Dados de nível universitário. O projeto aborda o desafio do mundo real da logística de entrega de última milha (*last-mile delivery*), um problema crítico e computacionalmente intensivo na economia moderna.[1] Os estudantes irão implementar um sistema multicamadas que começa com a análise de dados geoespaciais do mundo real, a construção de um grafo navegável, a implementação e análise de algoritmos de busca de caminho fundamentais (Dijkstra e A*), e, finalmente, a aplicação de heurísticas para resolver uma versão com restrições do Problema de Roteamento de Veículos (VRP). O projeto está estruturado para uma equipa ágil de seis pessoas e coloca uma forte ênfase na aplicação prática de estruturas de dados fundamentais — incluindo filas de prioridade, filas padrão e pilhas — e na análise rigorosa da complexidade algorítmica (notação Big-O).

Parte I: Arquitetura do Sistema e Fundamentos de Dados

Esta parte detalha a engenharia de dados fundamental e os componentes de estrutura de dados do sistema OptiRota. O desafio principal é converter informações geográficas não

estruturadas do mundo real numa estrutura de dados de grafo limpa e computacionalmente eficiente, sobre a qual os nossos algoritmos possam operar.

1.1 Modelagem do Ambiente Urbano: De Dados Brutos a um Grafo Navegável

1.1.1 Fonte e Aquisição de Dados

O projeto utilizará o OpenStreetMap (OSM), uma base de dados de mapas globais colaborativa e de código aberto.[3] Os dados do OSM são ideais devido à sua alta qualidade para redes de ruas e acessibilidade.[3] Os dados para uma área urbana específica (por exemplo, um distrito da cidade) serão adquiridos a partir de extratos pré-embalados (por exemplo, do Geofabrik) ou utilizando uma API como a Overpass API para consultar uma caixa delimitadora (

bounding box) específica.[6]

1.1.2 Compreendendo o Modelo de Dados do OSM

Antes da análise (*parsing*), é crucial compreender os tipos de dados primitivos do OSM: **Nós** (*Nodes*), que são pontos com latitude/longitude; **Caminhos** (*Ways*), que são listas ordenadas de nós representando polilinhas como ruas ou polígonos como edifícios; e **Relações** (*Relations*).[8] O significado destes primitivos é definido por

Etiquetas (*Tags*) (por exemplo, highway=primary, oneway=yes), que são essenciais para filtrar e construir o grafo de roteamento.[4]

1.1.3 O Processo de Análise e Construção do Grafo

Esta é uma tarefa de processamento de dados não trivial. A equipa precisará de usar uma biblioteca (por exemplo, pyroutelib3 para Python, ou um analisador personalizado) para ler um ficheiro OSM (em formato .osm.pbf ou .xml) e convertê-lo num grafo em memória.[4]

- **Representação do Grafo:** O mapa da cidade será modelado como um grafo ponderado e direcionado, G = (V, E).
- **Vértices (Nós):** Cada nó do OSM que representa um cruzamento de ruas tornar-se-á um vértice no nosso grafo.[15]
- **Arestas:** Cada segmento de um caminho do OSM que liga dois nós de cruzamento tornar-se-á uma aresta direcionada (ou um par de arestas direcionadas para ruas de dois sentidos) no nosso grafo.[3]
- **Pesos das Arestas:** O "custo" de atravessar uma aresta. Inicialmente, este será a distância geográfica entre os dois nós. Isto requer a conversão de coordenadas de latitude/longitude para um plano cartesiano (por exemplo, usando a fórmula de Haversine ou uma projeção de mapa como UTM) para calcular distâncias métricas.[8] Pesos mais avançados poderiam incorporar o tempo médio de viagem com base no tipo de estrada (etiqueta
  highway).[3]

A conversão de dados do OSM num grafo utilizável não é apenas um pré-requisito, mas um desafio central de engenharia de dados do projeto. A qualidade e o desempenho da aplicação de roteamento final dependem direta e criticamente da eficiência e correção desta fase inicial de análise e modelagem. Um analisador ingénuo pode criar um grafo com milhões de nós a partir de um pequeno mapa da cidade, incluindo todos os pontos de um caminho, e não apenas os cruzamentos.[9] Este grafo inchado aumentaria drasticamente o espaço de busca para os algoritmos de busca de caminho, tornando o cálculo em tempo real inviável, impactando diretamente os termos

|V| e |E| nas fórmulas de complexidade Big-O.[21] Portanto, uma estratégia de análise inteligente que identifica corretamente apenas os verdadeiros pontos de decisão (cruzamentos) é uma otimização crucial. Isto implica uma cadeia causal: Qualidade do Analisador → Tamanho do Grafo → Desempenho do Algoritmo → Viabilidade do Sistema. Isto destaca que as estruturas de dados não são apenas sobre teoria; são sobre tornar os dados do mundo real computacionalmente tratáveis.

1.1.4 Lidando com as Complexidades da Rede Viária do Mundo Real

Um grafo simples é insuficiente. O modelo deve ter em conta as restrições do mundo real encontradas nos dados do OSM:

- **Ruas de Sentido Único:** A etiqueta oneway=yes dita que uma aresta deve ser criada apenas numa direção.[18]
- **Restrições de Viragem:** Modelos mais avançados podem representar restrições de viragem (por exemplo, "proibido virar à esquerda"). Isto pode ser modelado expandindo os nós de cruzamento ou usando uma representação de grafo de linha, embora para este projeto, reconhecer esta complexidade seja suficiente; a implementação é uma extensão opcional.[18]
- **Regras de Acesso:** Filtrar caminhos com base em etiquetas como access=private ou vehicle=no é crucial para construir um grafo realista para veículos de entrega.[4]

1.2 Estruturas de Dados Centrais no OptiRota

1.2.1 A Fila de Prioridade para Busca de Caminho

Esta é a estrutura de dados mais crítica para os algoritmos centrais do projeto. Tanto o Dijkstra como o A* dependem de uma fila de prioridade para gerir eficientemente a "fronteira" ou "conjunto aberto" de nós a visitar em seguida.[21]

- **Função:** Armazena os nós a serem avaliados, priorizados pelo seu custo (distância da origem no Dijkstra; distância + heurística no A*). A operação extract-min permite que o algoritmo selecione sempre de forma gulosa o nó mais promissor a explorar em seguida.[21]
- **Implementação:** Embora um array simples possa ser usado, isso leva a um mau desempenho ($O(|V|)$ para extract-min). Um **heap binário** é a implementação padrão e necessária para este projeto, oferecendo $O(\log V)$ para inserções e extrações, o que é fundamental para alcançar a complexidade eficiente de $O(|E|\log V)$ para o Dijkstra.[21]

A escolha da implementação para a fila de prioridade é o fator mais significativo que determina a escalabilidade prática dos algoritmos de busca de caminho. A consulta do

utilizador requer uma análise Big-O. A investigação indica claramente que a complexidade do Dijkstra varia de O(|V|2) com um array simples para O(|E|logV) com um heap binário/fila de prioridade.[21] Para um grafo de cidade realista com milhares de vértices e arestas, uma complexidade quadrática é computacionalmente proibitiva, enquanto uma complexidade

O(|E|logV) é viável. Portanto, o projeto não é apenas sobre *usar* uma fila de prioridade; é sobre compreender *por que* a implementação subjacente do heap é fundamentalmente o que torna o roteamento em larga escala possível. Isto cria uma ligação direta entre uma escolha de estrutura de dados de baixo nível (um heap) e uma capacidade de sistema de alto nível (roteamento num mapa à escala da cidade), uma demonstração poderosa da importância do material do curso.

1.2.2 A Fila para Processamento de Pedidos

Para simular um depósito de logística do mundo real, os pedidos de entrega recebidos serão geridos usando uma **Fila** padrão (Primeiro a Entrar, Primeiro a Sair - FIFO).

- **Função:** À medida que novos pedidos de entrega são criados (por exemplo, lidos de um ficheiro), eles são colocados na fila (enqueue). O solucionador de VRP irá retirar (dequeue) os pedidos desta estrutura para construir uma lista de entregas para o próximo cálculo de roteamento. Isto modela um processamento justo e cronológico dos pedidos dos clientes.

1.2.3 A Pilha para Reconstrução do Caminho

Os algoritmos de busca de caminho normalmente encontram o objetivo e um conjunto de ponteiros de "predecessor" (ou seja, para cada nó, qual o nó que veio antes dele no caminho mais curto).[23] Uma

Pilha (Último a Entrar, Primeiro a Sair - LIFO) é a estrutura de dados canónica para inverter esta cadeia para produzir o caminho final e legível por humanos.

- **Função:** Começando no nó de destino, o algoritmo irá percorrer para trás usando os ponteiros de predecessor, colocando (pushing) cada nó na pilha até que o nó de início seja alcançado. Em seguida, retirar (popping) da pilha irá produzir o caminho na ordem

correta de A para B.

Parte II: O Núcleo Algorítmico: Busca de Caminho e Otimização Multi-paragem

Esta parte passa da representação de dados estática para a lógica dinâmica que produz as rotas ótimas. Abrange tanto algoritmos de caminho mais curto entre um par de pontos como o mais complexo problema de roteamento de veículos com múltiplas paragens.

2.1 Busca de Caminho Fundamental: Dijkstra vs. A*

2.1.1 Algoritmo de Dijkstra

Este algoritmo servirá como linha de base. Ele encontra o caminho mais curto de um único nó de origem para todos os outros nós num grafo com pesos de aresta não negativos.[21]

- **Mecanismo:** Funciona selecionando iterativamente o nó não visitado com a menor distância conhecida (usando a fila de prioridade), relaxando os seus vizinhos (atualizando as suas distâncias se um caminho mais curto for encontrado) e marcando-o como visitado.[21] Explora uniformemente para fora a partir da origem, como uma frente de onda.[21]
- **Aplicação:** Útil quando o destino é desconhecido ou quando são necessários caminhos para múltiplos destinos potenciais a partir de uma única origem.[29] É um algoritmo central em protocolos de roteamento de rede como o OSPF.[29]

2.1.2 Algoritmo de Busca A*

O A* é uma extensão do Dijkstra que otimiza para um único destino conhecido. É um algoritmo de busca informada.[24]

- **Mecanismo:** O A* modifica o valor de prioridade dos nós na fila de prioridade. A prioridade não é apenas o custo desde o início (g(n)), mas a soma desse custo e uma estimativa **heurística** do custo até ao objetivo (h(n)). O custo total é f(n) = g(n) + h(n).[32] Esta heurística "guia" a busca em direção ao destino, podando grandes partes do grafo que o Dijkstra exploraria desnecessariamente.[24]
- **A Função Heurística (h(n)):** Para este projeto, a heurística será a distância Euclidiana ou "em linha reta" do nó atual até ao nó de destino.
- **Admissibilidade:** Para que o A* garanta o caminho ótimo (mais curto), a heurística deve ser **admissível**, o que significa que nunca *superestima* o custo real para o objetivo.[24] A distância em linha reta num mapa é sempre admissível porque é o caminho mais curto possível entre dois pontos.

2.1.3 Análise de Complexidade Algorítmica (Big-O)

Este é um componente obrigatório do projeto.

- **Dijkstra:**
  - *Pior/Médio Caso:* O(|V|^2) com um array; O(|E| + |V|\log) ou simplesmente O(|E|\log|V|) para grafos conexos usando uma fila de prioridade de heap binário.[21]
  - *Melhor Caso:* Se o grafo estiver estruturado de forma que o algoritmo alcance o destino rapidamente, a complexidade permanece ligada ao número de arestas e vértices explorados, pelo que não muda significativamente do caso médio em termos de Big-O.
- *A:**
  - *Pior Caso:* No pior caso (por exemplo, com uma heurística fraca ou um grafo tipo labirinto que o força a explorar tudo), a complexidade do A* é semelhante à do Dijkstra, O(|E|\log|V|).[24]
  - *Médio/Melhor Caso:* O desempenho prático é a sua principal vantagem. Com uma boa heurística, o A* explora muito menos nós do que o Dijkstra, tornando-o significativamente mais rápido na prática, embora o seu limite de complexidade no pior caso permaneça o mesmo.[24]

Característica	Algoritmo de Dijkstra	Algoritmo de Busca A*

Estratégia	Busca não informada, gulosa	Busca informada, best-first
Função de Custo	g(n) (custo real da origem ao nó n)	f(n) = g(n) + h(n) (h(n) é a heurística para o destino)
Optimalidade	Garantida para pesos de aresta não negativos	Garantida se a heurística h(n) for admissível
Complexidade (Pior Caso)	O(	E
Vantagem Principal	Encontra o caminho mais curto para *todos* os nós a partir da origem	Significativamente mais rápido para encontrar um caminho para um *único* destino
Desvantagem Principal	Explora "cegamente" em todas as direções, ineficiente para um único destino	O desempenho depende da qualidade da heurística; requer um destino conhecido

2.2 O Desafio Avançado: O Problema de Roteamento de Veículos (VRP)

2.2.1 Definição do Problema

O VRP é o problema do mundo real de encontrar o conjunto ótimo de rotas para uma frota de veículos servir um conjunto de clientes.[36] É uma generalização do famoso Problema do

Caixeiro Viajante (TSP) e é NP-difícil, o que significa que não existe um algoritmo eficiente (em tempo polinomial) para encontrar a solução ótima para instâncias grandes.[38]

2.2.2 Âmbito do Projeto - VRP com Restrições (VRPTW & CVRP)

Este projeto irá implementar um solucionador baseado em heurísticas para um VRP com duas restrições práticas comuns:

- **VRP com Capacidade (CVRP):** Cada veículo tem uma capacidade máxima (por exemplo, peso ou volume de pacotes) que não pode ser excedida.[40]
- **VRP com Janelas de Tempo (VRPTW):** Cada cliente deve ser visitado dentro de uma janela de tempo específica (por exemplo, entre as 9h e as 11h).[36]

2.2.3 Abordagem de Solução Heurística

Como encontrar a solução ótima é inviável, a equipa irá implementar uma **heurística construtiva**. Uma boa escolha é a **Heurística do Vizinho Mais Próximo com Inserção**:

1. **Pré-cálculo:** Usar o algoritmo A* implementado para calcular uma matriz de distâncias/tempos de viagem entre todos os locais dos clientes e o depósito.
2. **Inicialização:** Iniciar uma nova rota para um veículo no depósito.
3. **Iteração:** A partir da localização atual, viajar para o cliente não visitado mais próximo que seja viável (não viole as restrições de capacidade ou janela de tempo).
4. **Inserção:** Se nenhum vizinho mais próximo for viável, tentar inserir os clientes restantes nas rotas existentes na posição que cause o menor aumento no tempo total de viagem, respeitando todas as restrições.
5. **Terminação:** Repetir até que todos os clientes estejam roteados ou não se possam fazer mais inserções viáveis (exigindo um novo veículo/rota).

Esta abordagem revela a natureza hierárquica dos problemas de otimização do mundo real. O problema "básico" do curso de estruturas de dados (caminho mais curto) é, na verdade, um subproblema ou um bloco de construção para um problema muito mais difícil e prático (VRP). Para resolver o VRP, é necessário saber o tempo de viagem entre quaisquer dois pontos (clientes). Este tempo de viagem é calculado executando um algoritmo de caminho mais curto no grafo da rede viária subjacente. Isto significa que o algoritmo A* desenvolvido na

Secção 2.1 não é um fim em si mesmo; é um serviço ou uma ferramenta que o solucionador de VRP irá chamar, potencialmente centenas de vezes, para construir a sua matriz de distâncias. Isto revela um conceito crucial de arquitetura de software: a **separação de preocupações**. O "Módulo de Busca de Caminho" (A*) é distinto do "Módulo de Sequenciamento" (heurística de VRP). Este design modular é um princípio chave da boa engenharia de software e uma lição valiosa para além dos próprios algoritmos.

Parte III: Execução do Projeto e Colaboração da Equipe

Esta parte fornece a estrutura de gestão de projetos e garantia de qualidade, traduzindo a especificação técnica num plano acionável para a equipa de estudantes.

3.1 Estrutura da Equipa Ágil e Responsabilidades dos Papéis

O projeto será gerido usando uma metodologia Ágil leve (por exemplo, sprints tipo Scrum). Esta estrutura é omnipresente na indústria de software e proporciona uma valiosa experiência prática.

- **Product Owner (PO) (1 pessoa):**
  - Responsabilidades: É o dono do "product backlog". Define e prioriza as funcionalidades como histórias de utilizador (por exemplo, "Como desenvolvedor, quero analisar dados do OSM para um grafo para poder executar algoritmos de busca de caminho."). Atua como o árbitro final na implementação de funcionalidades, garantindo que o projeto cumpre todos os requisitos do curso.
- **Project Manager (PM) / Scrum Master (1 pessoa):**
  - Responsabilidades: Facilita o planeamento dos sprints, as reuniões diárias (*daily stand-ups*) e as revisões dos sprints. Acompanha o progresso, identifica e ajuda a remover impedimentos (*blockers*), e garante que a equipa adere ao cronograma do projeto.
- **Desenvolvedores (Devs) (3 pessoas):**
  - Responsabilidades: Implementam os componentes técnicos centrais. Esta equipa pode subdividir as tarefas: um desenvolvedor na análise de dados/construção do

grafo, um no Dijkstra/A* e na fila de prioridade, e um na heurística de VRP e lógica de restrições.

- **Engenheiro de Garantia de Qualidade (QA) (1 pessoa):**
  - Responsabilidades: Este é um papel crítico que vai além da simples depuração. O engenheiro de QA irá projetar e implementar um conjunto de testes abrangente. Isto inclui testes unitários para estruturas de dados, testes de integração para algoritmos e, mais importante, **benchmarking de desempenho** e **validação algorítmica** (ver Secção 3.3).

Módulo/Fase do Projeto	Product Owner (PO)	Project Manager (PM)	Desenvolvedores (Devs)	Engenheiro de QA
Módulo 1: Análise de Dados e Modelo de Grafo	Define os critérios de aceitação para um grafo válido	Coordena a aquisição de dados e define o cronograma	**Responsável** pela implementação do parser e da construção do grafo	Define os casos de teste para a estrutura do grafo
Módulo 2: Fila de Prioridade e Dijkstra	Prioriza a implementação do algoritmo base	Remove bloqueios técnicos	**Responsável** pela implementação da Fila de Prioridade (Heap) e do Dijkstra	**Responsável** pelos testes unitários e de integração do Dijkstra
*Módulo 3: A e Heurística**	Confirma que a heurística cumpre os requisitos de admissibilidade	Facilita a discussão sobre a escolha da heurística	**Responsável** pela implementação do A* e da função heurística	**Responsável** por testar a optimalidade e o desempenho comparativo do A*
Módulo 4: Heurísticas de VRP e Restrições	Define as restrições (CVRP, VRPTW) a	Gere a complexidade da integração das restrições	**Responsável** pela implementação da heurística	**Responsável** por criar testes que validem o cumprimento

	serem implementadas		de VRP e da lógica de restrições	das restrições
Módulo 5: Testes e Benchmarking	Valida que os resultados do benchmark estão alinhados com os objetivos	Garante que a equipa tem tempo e recursos para os testes	Apoia o QA na execução e depuração dos benchmarks	**Responsável** por executar os benchmarks de desempenho e analisar os resultados
Relatório Final e Apresentação	Revê o relatório final para garantir que todos os requisitos foram cumpridos	**Responsável** pela coordenação da escrita do relatório e da preparação da apresentação	Contribui com as secções técnicas do relatório	Fornece dados e gráficos da análise de desempenho para o relatório

3.2 Roteiro de Desenvolvimento (Plano de Sprints)

Um plano sugerido de cinco sprints, com cada sprint a durar 1-2 semanas.

- **Sprint 1: Fundação.** Objetivo: Ingerir dados do OSM e construir um grafo direcionado, ponderado e válido. *Entregável: Um programa executável que recebe um ficheiro OSM e produz um objeto de grafo que pode ser inspecionado.*
- **Sprint 2: Busca de Caminho Base.** Objetivo: Implementar uma Fila de Prioridade baseada em heap binário e o algoritmo de Dijkstra. *Entregável: Uma função que encontra o caminho mais curto entre dois nós no grafo.*
- **Sprint 3: Busca de Caminho Otimizada.** Objetivo: Implementar a busca A* com uma heurística de distância Euclidiana. *Entregável: Uma segunda função de busca de caminho e um teste comparativo que mostra que explora menos nós do que o Dijkstra*

em média.

- **Sprint 4: Roteamento Avançado.** Objetivo: Implementar a heurística de VRP (por exemplo, Vizinho Mais Próximo com Inserção) usando o algoritmo A* para cálculos de distância. Implementar as restrições de CVRP e VRPTW. *Entregável: Uma função que recebe uma lista de entregas e produz um conjunto de rotas de veículos válidas*.
- **Sprint 5: Análise e Finalização.** Objetivo: Concluir todos os testes, executar benchmarks de desempenho, analisar os resultados e escrever o relatório final do projeto e a apresentação.

3.3 Garantia de Qualidade e Benchmarking de Desempenho

O papel do QA transforma o projeto de uma simples tarefa de codificação "fazer funcionar" numa experiência científica. Força os estudantes a confrontar as implicações do mundo real de conceitos teóricos como a notação Big-O e a admissibilidade da heurística. Qualquer estudante pode copiar uma implementação de Dijkstra online. No entanto, ser capaz de *provar* as suas características de desempenho empiricamente requer uma compreensão mais profunda. Liga a matemática abstrata da análise de complexidade à realidade concreta de um cronómetro. Da mesma forma, demonstrar o modo de falha do A* com uma heurística inadequada mostra um domínio do conceito para além da simples implementação do caso de sucesso. Isto eleva significativamente o rigor académico do projeto e proporciona uma experiência de aprendizagem muito mais profunda, tornando o papel do QA não apenas um "testador", mas um "cientista da computação experimental".

3.3.1 Teste de Correção Algorítmica (Papel do QA)

- **Testes Unitários:** Testar as implementações da Fila de Prioridade (propriedades do heap), Pilha e Fila de forma independente.
- **Testes de Integração:** Criar pequenos grafos de teste feitos à mão para verificar os algoritmos de busca de caminho. Estes grafos devem incluir casos extremos como:
  - Componentes desconexos (o caminho não deve ser encontrado).
  - Grafos com ciclos.
  - Ruas de sentido único.
  - Um caso de teste com uma heurística deliberadamente **inadmissível** para o A* para provar que ele retorna um caminho subótimo, demonstrando uma compreensão

profunda da teoria.[24]

3.3.2 Análise de Desempenho Empírica (Papel do QA)

Este é um componente académico chave para validar a análise teórica de Big-O.

- **Metodologia:** O engenheiro de QA irá gerar ou encontrar grafos de vários tamanhos (por exemplo, 100, 1.000, 5.000, 10.000 nós). Irá executar o Dijkstra e o A* nestes grafos para um número definido de pares de início/fim aleatórios e medir o tempo médio de execução.
- **Entregável:** Uma secção no relatório final com gráficos de **Tempo de Execução vs. Tamanho do Grafo (V+E)**. Estes gráficos devem confirmar visualmente as curvas de complexidade teóricas (por exemplo, a curva O(|E|\log|V|) deve parecer quase linearítmica, não quadrática). Isto fornece uma prova tangível dos ganhos de eficiência resultantes do uso das estruturas de dados corretas.

3.3.3 Qualidade da Solução do VRP

Como a solução do VRP é heurística, a sua "correção" reside na viabilidade. Os testes de QA devem garantir que nenhuma rota gerada viola as restrições de capacidade (CVRP) ou de janela de tempo (VRPTW).

Referências citadas

1. Understanding Last-Mile Delivery Challenges and Solutions - Staci Americas, acessado em setembro 1, 2025, https://www.staciamericas.com/blog/understanding-last-mile-delivery-challenges-and-solutions
2. Last-Mile Logistics: Challenges and Solutions - KNAPP, acessado em setembro 1, 2025, https://www.knapp.com/en/insights/blog/last-mile-logistics-challenges-and-solutions/
3. Route Planning with OpenStreetMap - Jakob Miksch, acessado em setembro 1, 2025, https://jakobmiksch.eu/post/openstreetmap_routing/
4. Routing - OpenStreetMap Wiki, acessado em setembro 1, 2025,

https://wiki.openstreetmap.org/wiki/Routing
5. mplewis/osm-pathfinding: Apply A* to real life for once. Final project for UMN CSCI 4511W with @JoeSelvik and @kevana. - GitHub, acessado em setembro 1, 2025, https://github.com/mplewis/osm-pathfinding
6. MKuranowski/pyroutelib3: Simple routing over OpenStreetMap data - GitHub, acessado em setembro 1, 2025, https://github.com/MKuranowski/pyroutelib3
7. OpenStreetMap data analysis: how to parse the data with Python? - Oslandia, acessado em setembro 1, 2025, https://oslandia.com/en/2017/07/03/openstreetmap-data-analysis-how-to-parse-the-data-with-python/
8. Urban Graph Networks | Towards Data Science, acessado em setembro 1, 2025, https://towardsdatascience.com/urban-graph-networks-3723ac92c926
9. Using OpenStreetMap data - Algorithms, acessado em setembro 1, 2025, https://algo.win.tue.nl/tutorials/openstreetmap/
10. Urban Graph Networks - Towards Data Science, acessado em setembro 1, 2025, https://towardsdatascience.com/urban-graph-networks-3723ac92c926/
11. Tutorials about parsing osm.pbf?, acessado em setembro 1, 2025, https://help.openstreetmap.org/questions/25685/tutorials-about-parsing-osmpbf/
12. Create my own navigation/routing system [closed] - GIS Stack Exchange, acessado em setembro 1, 2025, https://gis.stackexchange.com/questions/137880/create-my-own-navigation-routing-system
13. Routing Engine Using OpenStreetMap Data - Stack Overflow, acessado em setembro 1, 2025, https://stackoverflow.com/questions/14428891/routing-engine-using-openstreetmap-data
14. A* Pathfinding in Go with Open Street Map data | by 최우진 - Medium, acessado em setembro 1, 2025, https://medium.com/@cdw1432m/a-pathfinding-in-go-with-open-street-map-data-f924958edff7
15. Good data structure for storing graph-like map : r/gamedev - Reddit, acessado em setembro 1, 2025, https://www.reddit.com/r/gamedev/comments/f18md9/good_data_structure_for_storing_graphlike_map/
16. Maps of real-world cities in graph representation (with nodes and edges) - Ignacio Arnaldo - GitHub Pages, acessado em setembro 1, 2025, http://ignacioarnaldo.github.io/OpenStreetMap2Graph/
17. City exploration algorithm - Software Engineering Stack Exchange, acessado em setembro 1, 2025, https://softwareengineering.stackexchange.com/questions/253367/city-exploration-algorithm

18. Road Networks Explained: Turning Geography into a Navigable Graph - Rico Fritzsche, acessado em setembro 1, 2025, https://ricofritzsche.me/road-networks-explained-turning-geography-into-a-navigable-graph/
19. Optimal Routing in Urban Road Networks: A Graph-Based Approach Using Dijkstra's Algorithm - MDPI, acessado em setembro 1, 2025, https://www.mdpi.com/2076-3417/15/8/4162
20. What algorithms compute directions from point A to point B on a map? - Stack Overflow, acessado em setembro 1, 2025, https://stackoverflow.com/questions/430142/what-algorithms-compute-directions-from-point-a-to-point-b-on-a-map
21. Dijkstra's algorithm - Wikipedia, acessado em setembro 1, 2025, https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
22. Modeling Costs of Turns in Route Planning, acessado em setembro 1, 2025, https://geo.fsv.cvut.cz/data/2013/pin2/d/dokumentace/line_graph_teorie.pdf
23. Path algorithm - Valhalla Docs, acessado em setembro 1, 2025, https://valhalla.github.io/valhalla/thor/path-algorithm/
24. Dijkstra vs. A* – Pathfinding | Baeldung on Computer Science, acessado em setembro 1, 2025, https://www.baeldung.com/cs/dijkstra-vs-a-pathfinding
25. Algorithms for Computing Routes on a Map - Baeldung, acessado em setembro 1, 2025, https://www.baeldung.com/cs/routes-optimal-on-map
26. How to implement Priority Queue - using Heap or Array? - GeeksforGeeks, acessado em setembro 1, 2025, https://www.geeksforgeeks.org/dsa/how-to-implement-priority-queue-using-heap-or-array/
27. Priority queue - Wikipedia, acessado em setembro 1, 2025, https://en.wikipedia.org/wiki/Priority_queue
28. pages.cs.wisc.edu, acessado em setembro 1, 2025, https://pages.cs.wisc.edu/~cs400/readings/Priority-Queues/#:~:text=A%20Priority%20Queue%20can%20be,data%20structure%20called%20a%20heap.
29. Dijkstra's Algorithm Explained: Comprehensive Guide to Shortest Paths - Upper Route Planner, acessado em setembro 1, 2025, https://www.upperinc.com/glossary/route-optimization/dijkstras-algorithm/
30. A summary of the routing algorithm and their optimization,performance - arXiv, acessado em setembro 1, 2025, https://arxiv.org/html/2402.15749v1
31. Introduction to A* - Stanford CS Theory, acessado em setembro 1, 2025, http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
32. algorithm - Difference and advantages between dijkstra & A star - Stack Overflow, acessado em setembro 1, 2025, https://stackoverflow.com/questions/13031462/difference-and-advantages-between-dijkstra-a-star
33. Classification of Routing Algorithms - GeeksforGeeks, acessado em setembro 1,

2025, https://www.geeksforgeeks.org/computer-networks/classification-of-routing-algorithms/
34. Dijkstra's algorithm vs. A* search. Is one better than the other? | by Rakshith Nagaraj, acessado em setembro 1, 2025, https://medium.com/@rakshith.nagaraj6/dijkstras-algorithm-vs-a-search-is-one-better-than-the-other-3c1f7a52a20d
35. www.upperinc.com, acessado em setembro 1, 2025, https://www.upperinc.com/glossary/route-optimization/dijkstras-algorithm/#:~:text=and%20end%20goal.-,When%20to%20use%3A,heuristic%20data%20about%20the%20goal.
36. Chapter 3 VEHICLE ROUTING PROBLEM WITH TIME WINDOWS, acessado em setembro 1, 2025, http://alvarestech.com/temp/vrptw/Vehicle%20Routing%20Problem%20with%20Time%20Windows.pdf
37. Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms - ResearchGate, acessado em setembro 1, 2025, https://www.researchgate.net/publication/220413310_Vehicle_Routing_Problem_with_Time_Windows_Part_I_Route_Construction_and_Local_Search_Algorithms
38. A Systematic Literature Review of Vehicle Routing Problems with Time Windows - MDPI, acessado em setembro 1, 2025, https://www.mdpi.com/2071-1050/15/15/12004
39. Vehicle Routing Problem with Time Windows to Minimize Total Completion Time in Home Healthcare Systems - MDPI, acessado em setembro 1, 2025, https://www.mdpi.com/2227-7390/11/23/4846
40. Capacity Constraints | OR-Tools | Google for Developers, acessado em setembro 1, 2025, https://developers.google.com/optimization/routing/cvrp
41. Vehicle routing problem: definition, challenges and solutions, acessado em setembro 1, 2025, https://antsroute.com/en/solutions/vehicle-routing-problem-challenges-solutions-and-practical-examples/

UNIMA
CENTRO UNIVERSITÁRIO DE MACEIÓ | Afya

RESUMO:

Objetivo Principal: Construir um sistema de otimização de rotas para um serviço de entregas urbanas. O projeto aplica conceitos teóricos de estruturas de dados e algoritmos a um problema prático e muito comum no mercado de trabalho.

O Que Fazer (Passo a Passo):

1. **Transformar um Mapa Real em um Grafo:**
    - **Ação:** Pegar dados de um mapa de verdade (usando o OpenStreetMap) e convertê-los em uma estrutura de grafo que o computador entenda.
    - **Como:**
        - **Nós (Vértices):** Serão os cruzamentos das ruas.
        - **Arestas:** Serão os trechos de rua que ligam os cruzamentos.
        - **Pesos:** O "custo" de cada aresta será a distância ou o tempo de viagem.
2. **Encontrar o Menor Caminho (De um Ponto A para um Ponto B):**
    - **Ação:** Implementar dois algoritmos clássicos para achar a rota mais curta entre dois pontos no grafo que vocês criaram.
    - **Algoritmos:**
        - **Dijkstra:** É o algoritmo base. Ele funciona, mas explora o mapa em todas as direções, o que pode ser lento.
        - **A* (A-Estrela):** É uma versão otimizada do Dijkstra. Ele usa uma "heurística" (uma estimativa inteligente da distância até o destino) para focar a busca e encontrar o caminho muito mais rápido na prática.
    - **Estrutura de Dados Chave:** Para que esses algoritmos sejam eficientes, é **obrigatório** o uso de uma **Fila de Prioridade** (implementada com um *heap*).
3. **Resolver o Problema Real (Múltiplas Entregas):**
    - **Ação:** Esta é a parte mais avançada. O sistema não vai apenas de A para B, mas deve planejar as melhores rotas para uma frota de veículos que precisa atender **vários clientes** em um dia. Isso é conhecido como o **Problema de Roteamento de Veículos (VRP)**.

- **Restrições do Mundo Real:** O sistema deve lidar com:
  - **Capacidade do Veículo (CVRP):** O total de pacotes em um veículo não pode exceder sua capacidade máxima.
  - **Janelas de Tempo (VRPTW):** Cada cliente deve ser atendido dentro de um horário específico (ex: das 9h às 11h).
- **Como Fazer:** Vocês usarão o algoritmo A* (criado no passo 2) como uma ferramenta para calcular as distâncias entre os clientes e, a partir daí, criar as rotas usando uma heurística (como a do "vizinho mais próximo").

Foco em Estruturas de Dados e Análise:

- **Estruturas Obrigatórias:**
  - **Fila de Prioridade (com Heap):** O coração da eficiência dos algoritmos de busca de caminho.
  - **Fila (FIFO):** Para gerenciar os pedidos de entrega na ordem em que chegam.
  - **Pilha (LIFO):** Para reconstruir o caminho final (do destino de volta para a origem) após o algoritmo terminar.
- **Análise de Complexidade (Big-O):** É fundamental que a equipe analise e apresente a complexidade de tempo (pior caso, caso médio e melhor caso) para os algoritmos de Dijkstra e A*.

Divisão Sugerida da Equipe (6 pessoas):

- **Product Owner (PO) (1 pessoa):** Define e prioriza as tarefas. É o "cliente" do projeto.
- **Project Manager (PM) (1 pessoa):** Organiza o time, controla o cronograma e remove impedimentos.
- **Desenvolvedores (3 pessoas):**
  - **Dev 1:** Focado na parte de dados (pegar o mapa do OSM e construir o grafo).
  - **Dev 2:** Focado nos algoritmos de busca (Dijkstra, A* e a Fila de Prioridade).
  - **Dev 3:** Focado no problema avançado (a lógica do VRP com capacidade e janelas de tempo).
- **Engenheiro de QA (1 pessoa):** Cria os testes para garantir que tudo funciona, valida os algoritmos e, mais importante, mede o tempo de execução para comprovar a análise de Big-O.

UNIMA
CENTRO UNIVERSITÁRIO DE MACEIÓ